A Dynamic MaxSAT-based Approach to Directed Feedback Vertex Sets*

Rafael Kiesel

Institute of Logic and Computation
TU Wien, Vienna, Austria
r.kiesel@tuwien.ac.at

André Schidler

Algorithms and Complexity Group
TU Wien, Vienna, Austria
aschidler@ac.tuwien.ac.at

Abstract

We propose a new approach to the *Directed Feedback Vertex Set Problem (DFVSP)*, where the input is a directed graph and the solution is a minimum set of vertices whose removal makes the graph acyclic.

Our approach, implemented in the solver DAGER, is based on two novel contributions: Firstly, we add a wide range of data reductions that are partially inspired by reductions for the similar vertex cover problem. For this, we give a theoretical basis for lifting reductions from vertex cover to DFVSP but also incorporate novel ideas into strictly more general and new DFVSP reductions.

Secondly, we propose dynamically encoding DFVSP in propositional logic using *cycle propagation* for improved performance. Cycle propagation builds on the idea that already a limited number of the constraints in a propositional encoding is usually sufficient for finding an optimal solution. Our algorithm, therefore, starts with a small number of constraints and cycle propagation adds additional constraints when necessary. We propose an efficient integration of cycle propagation into the workflow of MaxSAT solvers, further improving the performance of our algorithm.

Our extensive experimental evaluation shows that DAGER significantly outperforms the state-of-the-art solvers and that our data reductions alone directly solve many of the instances.

1 Introduction

The *Directed Feedback Vertex Set Problem (DFVSP)* is one of Karp's original 21 NP-complete problems [21] and has a wide range of applications such as for argumentation frameworks [10, 11], deadlock detection, program verification and VLSI chip design [33]. The problem is to find for a given directed graph a set of vertices—a Directed Feedback Vertex Set (DFVS)—such that every directed cycle uses at least one of the DFVS' vertices. In this paper, we consider the optimization version of the problem, where we search for a minimum DFVS of the graph.

While the problem is known to be fixed parameter tractable [8] in the size of the minimum DFVS, these algorithms do not help us in practice [15], where the size of a minimum DFVS can become prohibitively large. Other possible approaches include explicit branch-and-bound solvers [27], as well as encodings into a standard format for optimization problems, however, also these do not manage to solve all practically relevant instances [27].

We attack this problem from two directions. First, we use knowledge from the *Vertex Cover Problem (VCP)* where the key to success is the application of *data reductions*, which results in

---

a smaller graph that allows for easier solving. The reductions' transfer from VCP to DFVSP is possible, since, VCP is a special case of DFVSP. And, although many DFVSP instances do not match this special case, we give a theoretical basis that allows us to lift many data reductions from VCP to DFVSP. We use the fact that for these data reductions it is often sufficient if the DFVSP instance behaves *locally* like a VCP instance. We use this to lift several VCP data reductions to DFVS and, additionally, introduce several more general data reductions.

Our second advancement revisits the idea of using a general purpose optimization solver for DFVSP. We consider Maximum Satisfiability (MaxSAT) solvers as they build on top of Propositional Satisfiability (SAT) solvers, which are designed for efficiently handling binary decisions, in our case inclusion or exclusion of a vertex, and have seen tremendous advances in the last decade.

The problem we face, when encoding DFVSP in propositional logic, is that the resulting formula quickly becomes prohibitively large. A straightforward well-performing encoding has size cubic in the number of vertices [20]. Worse, the size of the encoding that is most suitable for our purposes can become exponential in the number of vertices: we list all cycles and state that any solution must contain at least one vertex from every cycle.

Fortunately, it is known that a subset of an instance's constraints is often sufficient for finding a feasible solution for the whole instance. This is formalized in approaches like *counter-example guided abstraction refinement (CEGAR)* [9]. CEGAR starts with a subset of the constraints, called an under-abstraction, and whenever the solver returns an infeasible solution, further constraints are added for the next solver run, until a feasible solution is obtained. We propose an even faster approach: by implementing *cycle propagation* directly into the MaxSAT solver, it is possible to immediately add necessary constraints whenever they are needed, instead of waiting for the solver to finish. The solver then immediately corrects its decision and never returns an infeasible solution.

Our implementation won PACE 2022 [31].

1.1 Contributions

Our main contributions are as follows:

1. We found a general condition that allows us to lift data reductions from the vertex cover problem to the DFVSP problem.

2. For many reductions, we went even further and provided non-trivial generalizations of vertex cover reductions that are applicable in a wide range of cases.

3. We show that implementing cycle propagation is highly beneficial for MaxSAT-based DFVSP-solving, especially since it is tightly integrated into the MaxSAT solver.

4. In terms of data reductions, our experimental evaluation reveals that our novel reductions can significantly reduce instance sizes, especially on structured instances.

5. In terms of cycle propagation, the experiments showed a significant speedup, which is even larger when the cycle propagation happens within the MaxSAT solver.

Overall, the theoretical innovations of this work together with the highly engineered MaxSAT solvers as a basis provide a new and highly efficient strategy of exactly solving DFVSP instances that significantly outperforms any currently available implementation.

1.2 Related Work

CEGAR has been successfully used for various other problems such as Hamiltonian cycles [35], graph coloring [17], propositional circumscription [18], SMT solving [7], QBF Solving [19], and encoding other PSPACE-hard problems [24, 32]. While CEGAR approaches are sometimes integrated inside the solver (e.g., for SMT solving [7]), the abstraction refinement is usually added as an extra layer on top of the (Max)SAT solver. The idea of generating the extra clauses during

the solving, before the solver returns the assignment, is facilitated using *propagators*, which allow for adding extra logic at different points of the solver's solving process. This has been successfully used for SMT solving [7] and dynamic symmetry breaking [22]. To our knowledge, cycle propagation (Section 5) is the first integrated approach for solving a combinatorial optimization problem.

Regarding DFVSP solving, there has been previous work on data reductions [23, 25, 26, 27], approximate solvers [12, 38], and exact solvers [5, 13, 16]. Especially for the latter there has been a large increase recently, since DFVSP was the topic of this years PACE Challenge [31]. Here, many approaches were also based on a CEGAR-like method, i.e., repeatedly refining a small set of constraints, but using integer linear programming solvers instead of MaxSAT solvers. Nevertheless, to the best of our knowledge both our data reductions as well as the idea of using cycle propagation *within* a solver for DFVSP are novel.

2 Preliminaries

(Di)Graphs We consider both undirected and directed graphs (digraphs). For a (di)graph G, we denote by $V(G)$ its vertices and by $E(G)$ its edges (or arcs if G is a digraph). Further, we denote an (undirected) edge between vertices u and v as $\{u, v\}$ and the arc from u to v as (u, v). If for an arc $(u, v) \in E(G)$ also the arc in the other direction is present, i.e., $(v, u) \in E(G)$, then we call it a bi-edge, and an arc (u, u) is called a *loop*. The neighbors of a vertex in a graph G are denoted by $N^G(v)$ and for a digraph we use $N_{pre}^G(v), N_{succ}^G(v), N_{bi}^G(v), N^G(v)$ for the set of predecessors, successors, their intersection and their union, respectively. If G is clear from the context, we may omit the superscript. In a (di)graph a (di)clique is a set of vertices S where each $v \in S$ satisfies $S \setminus \{v\} \subseteq N(v)$.

In the later sections we need some operations to modify (di)graphs. We define
- $G - S$ as the (di)graph obtained by removing a set S of vertices or edges from a (di)graph G. If S is a singleton set $\{v\}$ or $\{(u, v)\}$ we may omit the brackets and write $G - v$ or $G - (u, v)$.
- $G + S$ is defined analogously but adds vertices or edges.
- $G[S]$ as the induced sub(di)graph of G with respect to a set of vertices S, i.e., $V(G[S]) = S$ and $E(G[S]) = E(G) \cap S \times S$.
- $\Pi(G)$ for a digraph G as the graph with $V(\Pi(G)) = V(G)$ and $E(\Pi(G)) = \{\{u, v\} \mid (u, v), (v, u) \in E(G)\}$.

DFVSP We can now introduce the central problem addressed in this paper.

Definition 2.1 (Cycle, DFVS)**.** *Given a digraph G a path is a list of vertices $v_1, \ldots, v_n$ such that for $i = 1, \ldots, n - 1$ there exists an arc $(v_i, v_{i+1}) \in E(G)$. A cycle is a path $v_1, \ldots, v_n$ such that $v_1 = v_n$. Furthermore, a path (or cycle) is uncovered, if there is no cycle $v'_1, \ldots, v'_m$ such that $\{v'_i \mid i = 1, \ldots, m\} \subsetneq \{v_i \mid i = 1, \ldots, n\}$ and the length of a cycle is the number of distinct vertices in the cycle. $\mathcal{C}(G)$ refers to the set of all uncovered cycles in G.*

A Directed Feedback Vertex Set (DFVS) *of G is a set $D \subseteq V$ such that every cycle of G contains at least one vertex in D. A minimum DFVS is a DFVS of minimum cardinality. We denote by $DFVS(G)$ the minimum cardinality of any DFVS of G.*

An example of a DFVSP-instance with a solution is given in Figures 1a and 1d. As an example for covered and uncovered cycles, consider the cycle a, c, b, a. The cycle is covered, as vertices of both (uncovered) cycles a, b, a and c, b, c are proper subsets of $\{a, b, c\}$.

We introduce problems (Vertex Cover, Propositional Satisfiability) and relate them to DFVSP. Throughout this paper, we make this correspondence clearer by using similar names for corresponding objects, e.g., C_i for a clauses in propositional logic as they correspond to cycles.

Whenever each uncovered cycle has length 2 and using $\Pi(G)$, we can state DFVSP as follows:
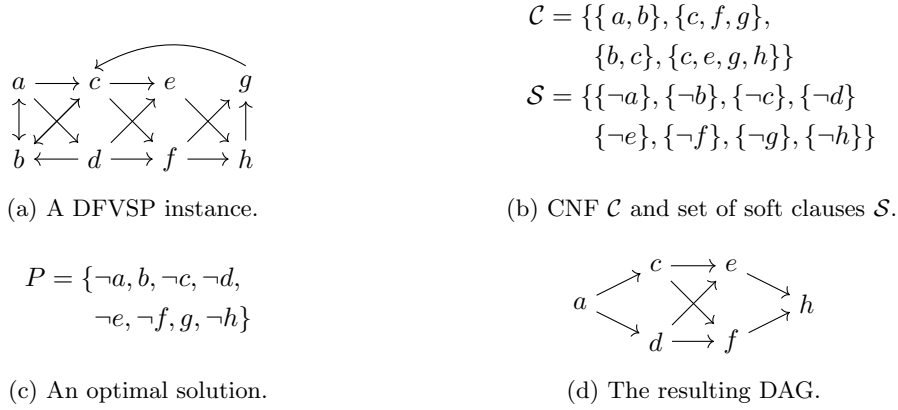
(a) A DFVSP instance.

$$\mathcal{C} = \{\{a, b\}, \{c, f, g\},$$
$$\{b, c\}, \{c, e, g, h\}\}$$
$$\mathcal{S} = \{\{\neg a\}, \{\neg b\}, \{\neg c\}, \{\neg d\}$$
$$\{\neg e\}, \{\neg f\}, \{\neg g\}, \{\neg h\}\}$$

(b) CNF $\mathcal{C}$ and set of soft clauses $\mathcal{S}$.

$$P = \{\neg a, b, \neg c, \neg d,$$
$$\neg e, \neg f, g, \neg h\}$$

(c) An optimal solution.

(d) The resulting DAG.

Figure 1: Example of a DFVSP instance, a solution, and its encoding as a MaxSAT instance.

Definition 2.2 (Vertex Cover). *Let G be an undirected graph. A minimum Vertex Cover (minimum VC) is a set $D \subseteq V(G)$, such that D is of minimum cardinality and for each edge $\{u, v\} \in E(G)$ it holds that $\{u, v\} \cap D \neq \emptyset$. $VC(H)$ denotes the minimum cardinality over all VCs of H.*

Propositional Logic and SAT solvers We give a brief introduction to propositional logic, SAT solvers, and MaxSAT solvers and refer the interested reader to [6] for more details.

We use propositional formulas in Conjunctive Normal Form (CNF). A CNF $\mathcal{C}$, defined for a set V of variables, is a finite conjunction of *clauses* C_i, where each clause consists of a finite disjunction of *literals* $\ell \in \{v, \neg v\}$ for some $v \in V$. We use the equivalency $\neg\neg v = v$ and say a literal ℓ is *negative* if it is of the form $\ell = \neg v$ and *positive* if $\ell = v$. Further, we denote clauses as a set of literals.

We represent truth assignments as a subset of $V \cup \{\neg v \mid v \in V\}$ that for any $v \in V$ do not contain both v and $\neg v$. Here, a positive literal indicates the value true, a negative literal the value false, and if a variable does not occur in the assignment, it isn't assigned a value (yet). We use the standard satisfaction relation and call an assignment that satisfies a formula a *model*.

A *propositional satisfiability (SAT) solver* takes as input a CNF and checks whether it has a model and returns one, if so. *MaxSAT* solvers additionally search for a model that optimizes an objective function. We use *partial MaxSAT*, where the solver takes two CNFs, called the *hard* and *soft* clauses. Then, a MaxSAT model is a model of the hard clauses that satisfies as many soft clauses as possible.

Consequently, we encode DFVSP as a MaxSAT instance by using $V(G)$ as the variables and adding for each uncovered cycle $v_1 \ldots v_n \in \mathcal{C}(G)$ a corresponding hard clause $\{v_1, \ldots, v_n\}$. Thus, if v_i is true it is in the DFVS. Accordingly, to achieve minimum cardinality we add a soft clause $\{\neg v\}$ for each $v \in V(G)$. A MaxSAT model then maximizes the elements that are not in the DFVS, effectively minimizing the elements that are in the solution. In Figures 1b and 1c we see an example of a MaxSAT encoding and the corresponding model.

3 Solver Architecture and Outline

We give a brief overview of our algorithm which also determines the structure of the paper. The general algorithm is shown in Algorithm 1.

The algorithm starts with performing general data reductions, discussed in Section 4 and then searches for a small set of short cycles.

Algorithm 1 The complete algorithm.

1: $G \leftarrow \text{Reduce}(G)$
2: $\mathcal{C}' \leftarrow \text{FindShortCycles}(G)$
3: **if** $\mathcal{C}' = \mathcal{C}(G)$ **then**
4: $\quad G, \mathcal{C}' \leftarrow \text{ReduceWithAllCycles}(G, \mathcal{C}')$
5: $D \leftarrow \text{DFVS\_MaxSAT}(G, \mathcal{C}')$
6: **return** D

Short cycles are cycles with a specified maximum length. Bounded depth first search can easily find these short cycles, as long as the bound, i.e., cycle length, is small enough. We discuss the associated limits on both the cycle length and the number of cycles in the implementation details in Section 6.

Should the set of short cycles represent all uncovered cycles, we can apply additional data reductions, which are also discussed in Section 4. We increase our chances of finding all uncovered cycles by ignoring the aforementioned limits and incrementally extending the maximum length of a cycle, as long as the number of new cycles we find decreases monotonically.

A MaxSAT solver is then used to solve the reduced instance. The set $\mathcal{C}' \subseteq \mathcal{C}(G)$ is given to the solver as the initial set of constraints. In case $\mathcal{C}'$ contains all uncovered cycles, the MaxSAT solver simply computes the minimum DFVS. When $\mathcal{C}'$ is not complete, the cycle propagation discussed in Section 5 ensures that the solver still returns a valid DFVS.

We evaluate how effective our solver is in Section 6.

4 Data Reductions[1]

Data reductions are the first step in our approach. They aim to shrink the input digraph in such a way that a minimum DFVS for the reduced digraph can easily be extended to a minimum DFVS for the original digraph. To this end, we apply a wide range of reduction rules. We give a list of all relevant reductions in Table 1.

4.1 DFVSP Reductions

For DFVSP there is already a wide range of rules by Levy and Low [26], Lemaic [25], Lin and Jou [27], which we use in an unmodified manner. For space reason, we do not repeat them here but refer to Appendix A for details.

Apart from this, we generalized DOME [27] from arcs dominated by length 2 paths to arcs dominated by arbitrary length paths:

Reduction 1 (DOME++)**.** *If there is an arc* $(v, u) \in E(G)$ *such that* $(u, v) \notin E(G)$ *and (i) every path that starts at* v *and ends at* u *uses a bi-edge or a vertex from* $N_{pre}(u) \setminus \{v\}$ *or (ii) every path that starts at* u *and ends at* v *uses a bi-edge or a vertex from* $N_{succ}(v) \setminus \{u\}$, *then replace* G *by* $G - (v, u)$.

Additionally, while enumerating all uncovered cycles is generally not feasible, due to their potentially exponential number, it is often possible in practice. This allows the following reduction.

Reduction 2 (ALLCYCLES)**.** *If there is an arc* $(v, u) \in E(G)$ *such that every cycle that visits* v *immediately after* u *is covered, then replace* G *by* $G - (v, u)$.

Theorem 4.1. *DOME++ and ALLCYCLES are sound if* G *is loop-free.*

Apart from the DFVSP reductions, we also lifted and generalized VCP reductions.

---

[1] Proofs for all theorems are in Appendix C

Name	Origin	Strictly Subsumed by
LOOP	[26]	-
IN0/1	[26]	INDICLIQUE
OUT0/1	[26]	OUTDICLIQUE
INDICLIQUE	[25]	-
OUTDICLIQUE	[25]	-
DICLIQUE-2	[25]	-
DICLIQUE-3	[25]	-
PIE	[27]	ALLCYCLES
DOME	[27]	ALLCYCLES
DOME++	-	ALLCYCLES
ALLCYCLES	-	-
CORE	[27]	IN/OUTDICLIQUE
"Reduction 2"	[36]	SUBSET (DFVSP)
SUBSET (VCP)	[36]	SUBSET (DFVSP)
SUBSET (DFVSP)	-	-
2FOLD	[37]	MANYFOLD (DFVSP)
"Reduction 4"	[36]	MANYFOLD (DFVSP)
"Reduction 5"	[36]	MANYFOLD (DFVSP)
"Reduction 7.2"	[36]	MANYFOLD (DFVSP)
MANYFOLD (VCP)	[14]	MANYFOLD (DFVSP)
MANYFOLD (DFVSP)	-	-
4PATH (VCP)	[14]	4PATH (DFVSP)
4PATH (DFVSP)	-	-
UNCONFINED (VCP)	[37]	UNCONFINED (DFVSP)
UNCONFINED (DFVSP)	-	-
3EMPTY	[36]	-
TWIN	[37]	3EMPTY + MANYFOLD (DFVSP)
FUNNEL	[37]	SUBSET + MANYFOLD (DFVSP)
DESK	[37]	-

Table 1: A summary of all used (and some unused) reduction rules, their origin, and the (currently) most general rule subsuming it.

4.2 VCP Reductions

As noted already in the preliminaries, VCP and DFVSP are related. We formalized this intuition as follows:

Lemma 4.1. *Let G be a digraph, then $S \subseteq V(G)$ is a DFVS if and only if*
- *S is a VC of $\Pi(G)$, and*
- *S is a DFVS of $G - E(\Pi(G))$.*

Thus, we can treat a DFVSP instance as a combination of a VCP instance $\Pi(G)$ and a smaller DFVSP instance without bi-edges. Hence, given sufficient preconditions, it suggests itself to apply VCP reductions to DFVSP. We derive these preconditions using boundary reductions similar to those of [14], which intuitively locally replace one part of a graph by another.

Definition 4.1 (Boundary VCP Reduction). *A boundary VCP reduction is a tuple $r = \langle H, H', B, c \rangle$, where H, H' are graphs, $B \subseteq V(H) \cap V(H')$ is a set of non-isolated vertices in both H and H', and $c \in \mathbb{Z}$, such that for all $X \subseteq B$ it holds that*

$$VC(H - X) = VC(H' - X) + c.$$

A reduction r is applicable *to G, if the vertices in B are (i) not isolated in G, (ii) an independent set in G, and (iii) the overlapping vertices, i.e., $V(G) \cap V(H) = B = V(G) \cap V(H')$.*

Example 1. *An example of a boundary reduction $r = \langle H, H', B, c \rangle$ is given by the boundary $B = \{b_1, b_2\}$, size difference $c = 1$, and the graphs*

$$H = \quad b_1 \text{ --- } v_1 \text{ --- } v_2 \text{ --- } b_2 \qquad H' = \quad b_1 \text{ --- } b_2$$

r is a boundary reduction, since for $X \subseteq \{b_1, b_2\}, X \neq \emptyset$, there is always still the edge between v_1 and v_2 in $H - X$, which means that $VC(H - X) = 1$, whereas $H' - X$ is edgeless. On the other hand, if $X = \emptyset$, then $VC(H - X) = VC(H) = 2$ and $VC(H' - X) = VC(H') = 1$.

Importantly, applicability guarantees soundness and the desired locality property:

Theorem 4.2. *For every graph G such that $\langle H, H', B, c \rangle$ is applicable it holds that for every minimum VC S of $G + V(H) + E(H)$ there is a minimum VC S' of $G + V(H') + E(H')$ such that $|S| = |S'| + c$, and $S \cap V(G) = S' \cap V(G)$.*

Not only does this theorem guarantee that the size of a minimum VC changes by c, it also tells us that the modification only has local effects on the minimum VCs. This locality is particularly interesting for lifting VCP reductions to DFVSP, as it allows their application when a digraph "locally behaves like a VCP instance". This intuition is formalized in the following theorem.

Theorem 4.3. *Let G be a digraph, $r = \langle H, H', B, c \rangle$ be a boundary VCP reduction and $\Pi(G) = G' + V(H) + E(H)$ such that r is applicable to G'. If*
(i) all edges incident in G to any $v \in V(H) \setminus B$ are bi-edges and
(ii) for every arc $(u, w) \in E(G)$ at least one of the following holds:
(ii.a) (u, w) is a bi-edge or
(ii.b) $|\{u, w\} \cap B| \leq 1$ then

$$DFVS(G) = DFVS(G^*) + c,$$

where G^* is given by

$$\begin{aligned} G \quad &- (V(H) \setminus B) - B \times B \\ &+ V(H') + \{ (u, v) \mid \{u, v\} \in E(H') \}. \end{aligned}$$

We can, thus, use many VCP reductions without modification by checking the preconditions of Theorem 4.3. For space reasons, we do not introduce VCP reductions that we strictly generalize to the DFVSP setting later on. They can, however, be found in Appendix B. This leaves only the following reduction, which we apply without any changes:

Reduction 3 (3EMPTY [36, 14]). *If there exists a vertex $v \in V(G)$ such that $|N(v)| = 3$ and $|E(G[N(v)])| = 0$, then replace G by*

$$G - v + \{\{a, b\}, \{b, c\}\} + \{a\} \times N(b) + \{b\} \times N(c) + \{c\} \times N(a).$$

4.3 Directed Versions of VCP Reductions

Some VCP reductions can be generalized to DFVSP, even when Theorem 4.3 does not apply. Note that all of the following reductions are strict generalizations of VCP reductions. I.e., when a digraph only has bi-edges, then each of the new DFVS reductions corresponds to a VCP reduction.

A simple example of this is the SUBSET reduction:

Reduction 4 (SUBSET). *If there exists $v, u \in V(G)$ such that $(v, u), (u, v) \in E(G)$, $N_{pre}(v) \subseteq N_{pre}(u) \cup \{u\}$, and $N_{succ}(v) \subseteq N_{succ}(u) \cup \{u\}$, then replace G by $G - u$.*

Theorem 4.4. *Let G be a digraph. After applying SUBSET to vertices v, u resulting in G', it holds that for every minimum DFVS S of G' the set $S \cup \{u\}$ is a minimum DFVS of G.*

Other reductions such as the MANYFOLD reduction, are more advanced.

Reduction 5 (MANYFOLD). *If there exists a vertex $v \in V(G)$ such that $N(v) = N_{bi}(v)$ and there is a partition (C_1, C_2) of $N(v)$, where*
- $|C_1| \geq |C_2|$,
- $G[C_i]$ *is a diclique for $i = 1, 2$,*
- M *is the set of non-arcs of $G[N(v)]$,*
- $(c, d) \in M$ *implies that either $(d, c) \in M$ and there is no uncovered path between c and d, or $(d, c) \notin M$ and every uncovered path from d to c uses the arc (d, c)*
- *for each $c_1 \in C_1$, there is exactly one $c_2 \in C_2$ (denoted $c_2(c_1)$) such that $(c_1, c_2) \in M$ or $(c_2, c_1) \in M$,*

then, replace G by

$$G - v - C_2$$
$$+ \bigcup_{c_1 \in C_1} \{c_1\} \times N_{succ}(c_2(c_1)) \cup N_{pre}(c_2(c_1)) \times \{c_1\}$$
$$- \bigcup_{c_1 \in C_1} (c_1, c_1).$$

We go over the conditions to explain their relevance.
- $N(v) = N_{bi}(v)$ needs to hold, to ensure that when a minimum DFVS S does not contain v, then it must be the case that $N(v)$ is a subset of S.
- For each $c_1 \in C_1$ there is exactly one $c_2 \in C_2$ with missing arc (c_1, c_2) or (c_2, c_1). This ensures that when c_1 is not in a minimum DFVS S, then either all vertices from C_2 are in S or only the uniquely determined vertex c_2 is not in S.
- The conditions on M ensure that when we perform the contraction, we only add cycles for which there exists a corresponding cycle in the original digraph.

Note that the conditions on the arcs in M are NP-hard to check. Later, we discuss alternative tractable and sufficient conditions. First we state soundness.

Theorem 4.5. *Let G be a loop-free digraph, such that PIE is not applicable and MANYFOLD is applicable to $v^* \in V(G)$ and G' be the graph obtained from G after MANYFOLD was applied on vertex v^*, then $DFVS(G) = DFVS(G') + |C_2|$ and given a minimum DFVS of G', we can in polynomial time compute a minimum DFVS of G.*
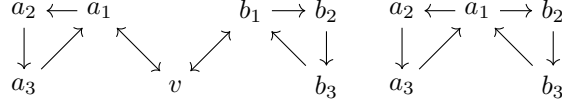
We need to check two possible conditions on the arcs in M. First, if $(d, c) \in E(G)$ we use straightness:

Definition 4.2 (Straightness). *Let G be a digraph and $(d, c) \in E(G)$. Then (d, c) is straight, if $(c, d) \notin E(G)$ and (i) every arc $(d, c') \in E(G)$ such that $c' \neq c$ is a bi-edge or (ii) every arc $(d', c) \in E(G)$ such that $d' \neq d$ is a bi-edge.*
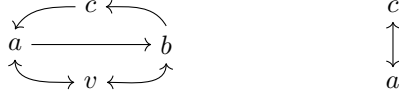
As desired, if the arc (d, c) is straight, then every uncovered path that contains d after c uses it. If, on the other hand, (d, c) is not in $E(G)$, we need to prohibit the existence of an uncovered path between c and d. Here, we give a sufficient condition using *Strongly Connected Components (SCCs)*. Recall that an SCC is a subset maximal set S of vertices such that for every combination v, u of vertices in S there is a (directed) path from u to v.

We consider for a digraph G the SCCs of $G - E(\Pi(G))$, i.e., G without bi-edges, and denote for a vertex $v \in V(G)$ by $\text{SCC}_{||}^G(v)$ the unique SCC containing it. If G is clear from the context, we may omit the superscript. Then, the PIE reduction [27] allows us to remove arcs between vertices u, v such that $\text{SCC}_{||}(v) \neq \text{SCC}_{||}(u)$. This entails that when $\text{SCC}_{||}(v) \neq \text{SCC}_{||}(u)$, every path between u and v uses a bi-edge and is thus covered.

Together these conditions give us a tractable way of guaranteeing applicability of MANY-FOLD regardless of whether both (c, d) and (d, c) are in M or whether only one of them is. Figure 2 shows example applications.

(a) A MANYFOLD reduction, where $\text{SCC}_{||}(a_1) = \{a_1, a_2, a_3\} \neq \{b_1, b_2, b_3\} = \text{SCC}_{||}(b_1)$.

(b) A MANYFOLD reduction, where the arc (a, b) is straight.

Figure 2: Two applications of the MANYFOLD reduction. In both cases, left is before, right is after.

Algorithm 2 Checks if v is unconfined in a digraph G.

1: **function** CHECKUNCONFINED(v, G)
2:      $A \leftarrow \{v\}$
3:      $N \leftarrow \{\, u \in V(G) \mid G[A \cup \{u\}] \text{ is cyclic}\,\}$
4:      $P \leftarrow \{u \in N \mid |N_{succ}(u) \cap A| + |N_{pred}(u) \cap A| = 2\}$
5:      **if** $P \neq \emptyset$ **then**
6:          $u \leftarrow \arg\min_{u' \in P} |N(u') \setminus (N \cup A)|$
7:          **if** $|N(u) \setminus (N \cup A)| = 0$ **then return** True
8:          **else if** $|N(u) \setminus (N \cup A)| = 1$ **then**
9:              $A \leftarrow A \cup \{u\}$
10:              **go to** 3
     return False

We note that it is not necessary to recompute SCCs at every step, since we can update them after each reduction in an approximate but safe manner and only recompute them periodically.

Apart from MANYFOLD, we can also generalize 4PATH in a similar manner, by exploiting the lack of uncovered paths between some of the involved vertices.

Reduction 6 (4PATH). *If there exists a vertex $v \in V(G)$ such that*
- $N(v) = N_{bi}(v) = \{a, b, c, d\}$,
- $E(G[N(v)]) = \{\, (a, b), (b, a), (b, c), (c, b), (c, d), (d, c)\,\}$,
- *and there is no uncovered path between any pair of vertices from $\{\, \{a, c\}, \{a, d\}, \{d, b\}\,\}$,*

then replace G by

$$G - v + \{(a, c), (c, a), (a, d), (d, a), (b, d), (d, b)\}$$
$$+ \{a, b\} \times N_{succ}(d) + N_{pred}(d) \times \{a, b\}$$
$$+ \{c, d\} \times N_{succ}(a) + N_{pred}(a) \times \{c, d\}.$$

Again, we practically ensure that every path is covered by requiring $\text{SCC}_{||}(a) \neq \text{SCC}_{||}(c), \text{SCC}_{||}(a) \neq \text{SCC}_{||}(d)$ and $\text{SCC}_{||}(d) \neq \text{SCC}_{||}(b)$. For either condition 4PATH is sound.

Theorem 4.6. *Let G be a digraph such that 4PATH is applicable to $v^* \in V(G)$ and G' be the graph obtained from G after 4PATH was applied to the vertex v^*, then*
- *$DFVS(G) = DFVS(G')$, and*
- *given a minimum DFVS of G', we can in polynomial time compute a minimum DFVS of G.*

Whereas the above reductions all capture fixed graph patterns, the applicability of the following one is determined by the iterative procedure in Algorithm 2.

Reduction 7 (UNCONFINED). *If there is a vertex $v \in V(G)$ such that* CHECKUNCONFINED(v, G) *returns True, replace G by $G - v$.*

When a vertex v is unconfined, this guarantees us that while there may be minimum DFVSs that do not contain v, there is at least one, which does.

Theorem 4.7. *Let G be a digraph. After applying UNCONFINED to vertex v resulting in G', it holds that for every minimum DFVS S of G' the set $S \cup \{v\}$ is a minimum DFVS of G.*

This concludes the data reductions that we use and brings us to the solving step.

5 MaxSAT Solver

We compute the minimum DFVS for the reduced instance using a MaxSAT solver. Recall from Section 2 that we add one disjunction per cycle containing exactly the variables corresponding to the vertices in the cycle. Since there is a 1:1 correspondence between vertices and variables, cycles and clauses, as well as model, and DFVS, we treat them synonymously in this section. We also refer to a DFVS candidate that does not break all cycles as *infeasible* and to a DFVS as a *feasible* solution.

Enumerating all cycles for a complete encoding is generally impossible in practice. A well-established technique that deals with this issue is CEGAR [9]. In the context of DFVSP, a CEGAR approach initially gives the solver a small, usually not comprehensive, set of uncovered cycles. Whenever the solver returns a solution that is not a valid DFVS, we add cycles that are are not broken by the infeasible solution. This is repeated until the solver returns a feasible solution. Very often, a comparatively small number of constraints, in our case cycles, is sufficient for finding a feasible solution.

The main drawback of this CEGAR approach is the computational overhead. While a solver's decision may quickly imply that the solution is infeasible, the solver may run long past that point until it returns a solution. Further, after an infeasible solution is returned, it is hard to determine which of the solver's decisions caused the infeasability. Lacking this knowledge, we have to add many cycles that are not necessary for guiding the solver towards a feasible solution.

We propose cycle propagation for improved performance. Cycle propagation adds the feasibility check directly into the MaxSAT solver's logic and adds the necessary cycles at exactly the point where the MaxSAT solver's decision would cause the solution to become infeasible.

We focus on *core-guided* MaxSAT solvers. Here, the MaxSAT solver implements the search for an optimal solution and calls the SAT solver repeatedly. For each SAT call, the MaxSAT solver extends the input CNF by extra clauses related to the search for an optimal solution [28]. We therefore add cycle propagation to the SAT solver, as the decisions that lead to infeasability are made here. In order to introduce cycle propagation, we first discuss the basics of CDCL, the algorithm used by most modern SAT solvers [29, 34].

5.1 Conflict Driven Clause Learning (CDCL)

We limit ourselves to a cursory discussion of CDCL that introduces the necessary concepts to understand cycle propagation. Remember from Section 2 that a SAT instance consists of variables V and clauses $\mathcal{C}$. The whole algorithm including cycle propagation is shown in Algorithm 3. Ignoring cycle propagation in the block starting at Line 13, the listing shows the basic CDCL algorithm.

CDCL incrementally extends a partial assignment D, assigning values to some of the variables, to a full assignment until it either obtains a model or knows that the formula is unsatisfiable. The algorithm only keeps unsatisfied clauses and removes satisfied ones (Line 5).

Conflicts occur when a clause C cannot be satisfied by any extension of D to a full assignment, because D contains the negation of C's literals, as is checked in Line 6. Here, two things can happen. If the conflict occurred without any prior decision, the set of clauses implies a conflict and the formula is unsatisfiable. Otherwise, CDCL learns a conflict clause: a clause based on

---
Algorithm 3 The modified CDCL algorithm.

1: **function** $\mathrm{DFVS\_CDCL}(G, \mathcal{C})$
2:      $D \leftarrow \emptyset$
3:      **while** $|\mathcal{C}| > 0$ **do**
4:          $\mathcal{C} \leftarrow \mathcal{C} \setminus \{\, C \in \mathcal{C} \mid D \cap C \neq \emptyset \,\}$
5:          $D' \leftarrow \{\, \neg \ell \mid \ell \in D \,\}$
6:          **if** $|\{\, C \in \mathcal{C} \mid |C \setminus D'| = 0 \,\}| > 0$ **then**
7:              **if** No Decisions **then return** FALSE
8:              $\mathcal{C} \leftarrow \mathcal{C} \cup \textsc{analyzeConflict}()$
9:              $\mathcal{C}, D \leftarrow \textsc{backtrack}()$
10:         **else if** $|\{\, C \in \mathcal{C} \mid |C \setminus D'| = 1 \,\}| > 0$ **then**
11:             $D \leftarrow \textsc{booleanClausePropagation}()$
12:         **else**
                   ▷ Cycle Propagation
13:             $V' = \{\, v \in V(G) \mid \neg v \in D \,\}$
14:             **if** $G[V']$ contains a cycle C **then**
15:                 $\mathcal{C} \leftarrow \mathcal{C} \cup C$
16:             **else**
17:                 $D \leftarrow D \cup \{\textsc{decideLiteral}()\}$
18:     **return** TRUE

---

the decisions that lead to the conflict and that prevents the solver from making the same set of decisions again. Afterwards, the solver backtracks, where it removes the corresponding literals from D and restores the corresponding removed clauses and literals to $\mathcal{C}$.

Boolean constraint propagation and *decisions* are used by CDCL to extend D. Boolean constraint propagation adds implied literals to D, where a literal is implied if there exists a clause where this literal is the only one remaining that can be satisfied by an extension of D, as is checked in Line 10. Decisions add a selected literal to D after exhaustively applying Boolean constraint propagation without a conflict as denoted in Line 17.

This description of CDCL is deliberately conceptual. Modern SAT and MaxSAT solvers are well engineered pieces of software that use sophisticated data structures and algorithms which are integral to their performance. Particularly conflict analysis, backtracking, and decisions have not been covered here. We refer the interested reader to [6] for more details.

With the knowledge of how CDCL works, we discuss the integration of cycle propagation next.

5.2   Cycle Propagation

Conflicts are a central concept in CDCL, as they signal the solver that a partial assignment is infeasible. Cycle propagation uses this mechanism to ensure that the solver stops as soon as D implies a cycle. We perform this check after Boolean constraint propagation in Line 13.

Cycles are only implied by negative literals, since negative literals indicate that a vertex remains in the graph. Hence, it is sufficient to check if the negative literals $V' = \{\, v \in V \mid \neg v \in D \,\}$ induce an acyclic graph, i.e., if $G[V']$ is acyclic. In case a cycle C is found, it is added as a clause to $\mathcal{C}$.

Adding C immediately causes a conflict, since by definition D contains the negation of C. Hence, we achieve our goal of immediately stopping the solver. Further, we add a single cycle and corresponding conflict clause, thereby minimizing the number of extra constraints. This usually guides the solver quicker to a feasible solution than adding several cycles after the solver returns an infeasible solution. Note that the solver with cycle propagation never returns an infeasible solution.

The acyclicity check is performed using a DAG implemented as a simple doubly linked data structure. Here, each vertex knows its predecessors, successors and has an order. The structure preserves two invariants: it is a DAG, and the order of a vertex is the maximum order over its predecessors plus one, or 0 if the vertex has no predecessors. Whenever a new vertex is inserted, its order is recursively propagated to the successors. Recursive calls are only necessary, if the propagated order plus one is larger than the successor's order. Should the propagation reach the inserted vertex, we have found a cycle and we remove the vertex, preserving the invariant that the structure is a DAG. Removal of a vertex requires recursively propagating the change to all successors whose ordering depends on the target vertex.

Cycle propagation is performed after Boolean constraint propagation for practical reasons. First, modern SAT solvers spend most of their time performing Boolean constraint propagation and can perform this task very fast. Checking for cycles after each change to D would, therefore, cause a considerable slowdown of the solver. Second, we keep track of the changes to the partial assignment in between cycle propagation runs. This allows us to perform the aforementioned modifications to our data structure in bulk, further speeding up the acyclicity check. With these considerations, the runtime percentage dedicated to cycle propagation shown in the profiler is in the low single digits as Boolean constraint propagation still takes up almost all of the runtime.

This concludes the conceptual description of our approach. Next, we discuss our empirical evaluation.

6   Experiments[2]

Instances   We use instances from the recent *PACE*, the argumentation framework competition *ICCMA*, and *random* graphs. The recent PACE provides 200 dedicated DFVSP instances.[3][4] The 137 ICCMA instances come from a recent argumentation framework competition[5], where we selected those instances with 50 to 1000 vertices. We also generated 1140 random instances using different parameterizations for the number of vertices and the probability an edge exists using the methodology of [38]. We generated 10 instances for each parameterization, which are reported as 114 instances, averaging the results over the respective 10 instances. The number of vertices ranges between 100 and 10000 and the average degree of a vertex varies from 2 to 50.

We preprocessed the instances by removing all self-loops, as the PACE instances met this requirement and the competition solvers were not able to deal with instances containing self-loops.

Implementation   We implemented the proposed algorithm in our solver DAGER[6]. Our implementation is based on the MaxSAT solver EvalMaxSAT [4], which uses Glucose 3 in the backend [3]. We chose EvalMaxSAT because it placed well in the 2021 MaxSAT evaluation[7] and the code base has no dependencies and can easily be modified and integrated.

We initially give up to 25000 short cycles with a maximum length of 4 to the MaxSAT solver. These limits have performed best overall. A lower maximum length does not find any cycles for some instances, while a higher maximum length seems to slow down the solver, as does a larger number of cycles.

Setup   Our implementation uses C++ and was compiled using gcc 7.5.0. We compared our solver to the second place PACE solver *grapa-java*[8], which uses a CEGAR-like approach together

---

with an integer linear programming solver and new data reductions.[9] As an additional baseline we also used a direct DFVSP encoding into *SAT*, based on the transitive closure encoding for acyclicity [20], using our data reductions for preprocessing.

We used a time limit of 30 minutes and a memory limit of 8 GB. The experiments were run on servers with two AMD EPYC 7402 CPUs, each with 24 cores running at 2.8 GHz, and using Ubuntu 18.04.

An instance counts as *solved*, if it was solved in all five runs, otherwise if it was solved in at least one run, it is counted as *partially solved*. The given values are averaged over all runs.

6.1 Solvers

Comparing DAGer's performance to that of other solvers, was the goal of our first experiment. The results, together with different configurations from the next experiment, are shown in Table 2 and as a cactus plot in Figure 3.

Solver	PACE		ICCMA		Random	
	S	P	S	P	S	P
DAGer	186	2	64	0	13	5
grapa-java	165	0	52	0	10	5
SAT	134	3	59	0	12	5
Configuration	S	P	S	P	S	P
No CP	180	2	62	0	11	6
No DR	151	6	63	1	13	5
No CP & DR	146	3	62	0	10	7

Table 2: Number of solved instances for different solvers and DAGER configurations. S and P show the number of solved and partially solved instances respectively.
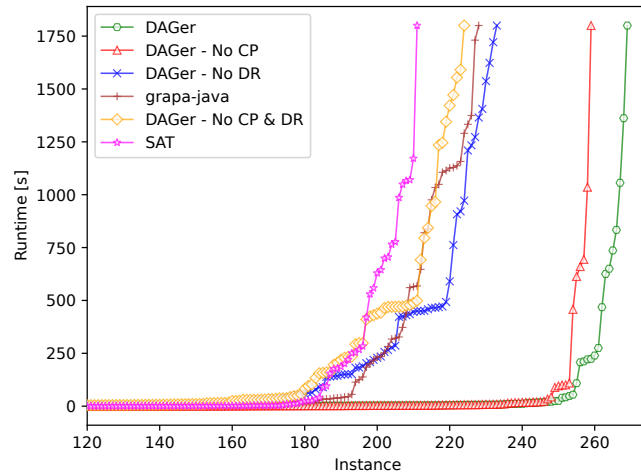

Figure 3: Cactus plot for different solvers and DAGER configurations.

---

[9]We tried to obtain further solvers for comparison. Unfortunately, for [5], we did not manage to get in contact with the authors, for [13] the source code is lost and the implementation of [23] did not contain an exact solver

DAGER performs better than both grapa-java and the SAT encoding for every instance group. Interestingly, the SAT encoding performs better than grapa-java on non-PACE instances. The cactus plot shows that instances are either very hard, or very easy, with very few solved instances having a high runtime.

DAGER excels on the PACE instances and solved almost half the ICCMA instances, but random instances seem to be hard for all solvers. We will examine this behavior for DAGER further in subsequent experiments.

6.2 Features

We measured the impact of our contributions by disabling cycle propagation (CP) or our new data reductions (DR). Table 2 shows the performance of these three additional configurations and Figure 3 shows the runtime behavior. Without cycle propagation, DAGER calls the MaxSAT solver incrementally and whenever the solution D is infeasible, we add disjoint cycles from $G - D$. Hence, our experiment tests precisely the benefit the integration into the solver.

Cycle propagation has a small impact in terms of the number of instances. While the number is small, the respective instances are hard and contain a very large number of uncovered cycles that we were unable to enumerate within the runtime. Without cycle propagation, DAGER solved 253 instances. The initial solution was infeasible for 132 of those instances, lazily generated clauses were not necessary for the remaining instances.

The impact of the data reduction depends strongly on the instance set. PACE instances are heavily reduced, but the impact on ICCMA and random instances is almost none. The reason for this is that our new reductions rely heavily on structural properties that are unlikely in randomized graphs. For ICCMA instances, the reason is different, which we will explore next.

Overall, without our contributions, DAGER would perform worse than grapa-java, but better than the SAT baseline, showing that the incremental approach is the more promising SAT approach for DFVSP.

6.3 Data Reductions

The effectiveness of the data reductions is not well represented by the number of instances the solving algorithm can solve, as in the future they might be beneficial for instances that are too hard for current solvers. Figure 4 shows how much all instances were reduced in size and offers some interesting insights. First, many instances, particularly ICCMA instances, are directly solved by our data reductions. The ICCMA instances seem to be either easily reducible, in which case they are also easy to solve, or they are hard to reduce and solve. Second, the result on random instances shows that the reductions become less effective with increasing density. Lastly, on most instances, the data reductions can significantly decrease the instance's size.

We also wanted to see how much benefit our computationally more expensive reductions have over the simple reductions proposed by Levy and Low [26]. Figure 5 shows that our data reductions do not have much benefit over the simple reductions on random instances. For PACE, instances the reductions are very useful, reducing the size of almost all instances, directly solving many of them. For the ICCMA instances, the reductions either solve the instance or are ineffective.

6.4 Instance Size

The potential correlation between the graph's size and the MaxSAT solver's ability to find a minimum DFVS, was the focus of our last experiment. Figure 6 shows which instances have been solved, partially solved, or remained unsolved, in relation to the number of vertices and density. While the number of uncovered cycles would also have been of interest, we could not enumerate
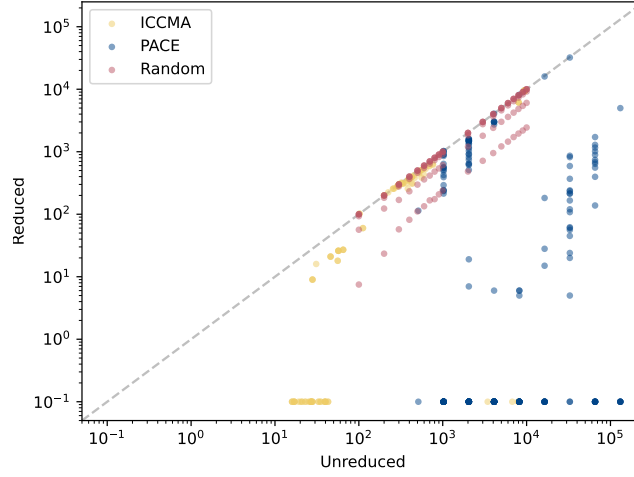
Figure 4: Comparison of graph sizes before and after applying data reductions. Due to the use of the logarithmic scale, we treat 0 as 0.1.
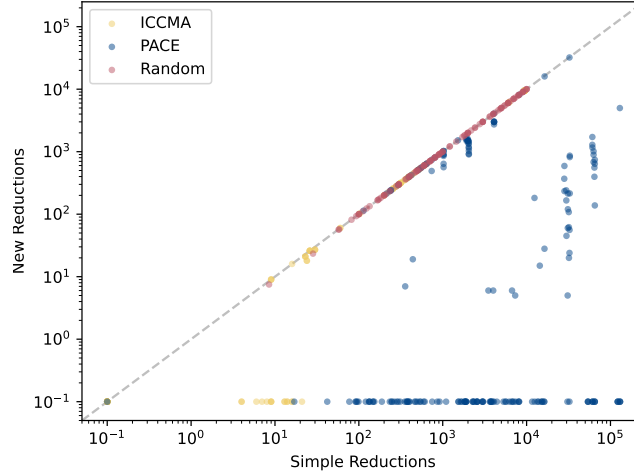

Figure 5: Comparison between using only simple reductions and using the data reductions we propose. Due to the use of the logarithmic scale, we treat 0 as 0.1.

them in a reasonable amount of time for hard instances. Since we were interested in the MaxSAT solver's performance, the figure uses the data from the reduced instances.

The figure shows that increased size and density indeed make the instance harder. Whereas for small graphs up to around 100 vertices the density does not matter much, this changes for larger graphs. The size limit for our approach seems to be around 10000 vertices, where the solver fails even for very sparse graphs.

Interestingly, at around 1000 vertices there is a cluster of instances with high density that the solver solved successfully. It seems that instances where almost the whole graph is part of a minimum DFVS are again easier to solve than mid-density instances.
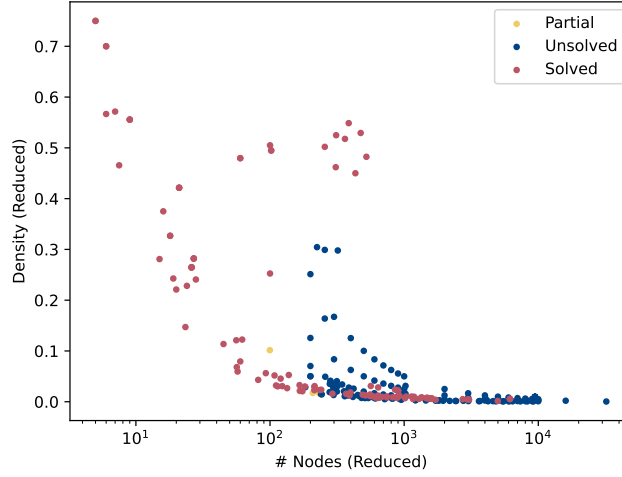
Figure 6: Solved instances in relation to the graph size and the density.

Random instances provide some more insight. For 100 vertices, DAGER solved almost all instances. The only exception is when the minimum DFVS size is around 50, here DAGER only managed to solve half the instances, hence, they seem to be harder. For the remaining instances, DAGER was not able to solve instances with average degree higher than 3, but managed to solve random instances with up to 1000 vertices.

6.5 Discussion

The results show that the data reductions perform particularly well on the PACE instances, where many instances have a high enough number of bi-edges. While still useful on the ICCMA instances, as there they solve many instances directly, they are not necessary, as the solver would have solved almost all of them.

Cycle propagation works in a complementary fashion to our data reductions. It works particularly well on instances that have few uncovered short cycles, but a large number of uncovered cycles. While not many of the instances in our instance sets fell into this category, cycle propagation helped to solve several hard instances.

In general, a CEGAR approach works well for DFVSP as even without cycle propagation and our data reductions, the solver outperformed the second best PACE solver on non-PACE instances.

Particularly challenging for all tested solvers are instances of high edge density, although there is a visible trend that indicates that instances with very high density could in turn become easier.

7 Conclusion

In this paper, we discussed our novel approach to DFVSP. Key features are new data reductions lifted from related problems. Apart from the reductions themselves, we also provided a theoretical basis that can be used to lift further reductions in the future. The other key feature is cycle propagation. While lazily extending the set of constraints to obtain a feasible solution with a limited set of constraints works well, we managed to solve several hard instances by integrating this extension directly into the MaxSAT solver.

16

There are more data reductions from VCP that we did not consider for DFVSP, since they are based on very non-local conditions or seem highly difficult to adapt to DFVSP such as the CROWN [1] or LP-based reduction [30]. We nevertheless hope to incorporate more reduction techniques. For local VCP reductions Theorem 4.3 is a strong methodological foundation for such a transfer, however, we hope to expand this and possible incorporate ideas from other related problems.

We have shown that cycle propagation works well in practice. We think that there are two avenues where we might further improve its performance: (i) there are several SAT solver details that might be used to further improve performance, particularly adapting inprocessing and decision heuristics to incorporate domain specific knowledge about DFVSP, and (ii) we used a core-guided MaxSAT solver for our implementation; it would be interested to see how well cycle propagation performs integrated into an implicit hitting set based MaxSAT solver.

A Standard DFVSP Reductions

Here, we use $G \circ v$ as the digraph, called the exclusion of v from a digraph G by letting $G \circ v :=$ $G - v + N_{succ}(v) \times N_{pre}(v)$. The most well-known reduction rules for DFVSP preprocessing are those of Levy and Low [26]:

Reduction 8 (LOOP). *If there exists $v \in V(G)$ such that $(v, v) \in E(G)$ replace G by $G - v$.*

Reduction 9 (IN0/1). *If there exists $v \in V(G)$ such that v has at most one incoming edge, replace G by $G \circ v$.*

Reduction 10 (OUT0/1). *If there exists $v \in V(G)$ such that v has at most one outgoing edge, replace G by $G \circ v$.*

The latter two rules were later subsumed by Lemaic [25], using the reductions.

Reduction 11 (INDICLIQUE). *If there exists $v \in V(G)$ such that the incoming edges of v form a diclique, replace G by $G \circ v$.*

Reduction 12 (OUTDICLIQUE). *If there exists $v \in V(G)$ such that the outgoing edges of v form a diclique, replace G by $G \circ v$.*

Apart from this, Lemaic introduced two new reductions

Reduction 13 (DICLIQUE-2). *If there exists $v \in V(G)$ whose neighbors can be partitioned into two disjoint cliques N_1, N_2 such that the bi-edges of v are a strict subset of N_1, replace G by $G \circ v$.*

Reduction 14 (DICLIQUE-3). *If there exists $v \in V(G)$ without bi-edges whose neighbors can be partitioned into three disjoint cliques $N_1, N_2, N3$, replace G by $G \circ v$.*

Note, that for soundness of all the above reductions it is necessary that LOOP is not applicable to v.

Furthermore, Lin and Jou introduced three further reductions that make use of bi-edges in the digraph.

Reduction 15 (PIE). *If there is an arc $(u, v) \in E(G)$ such that $(v, u) \notin E(G)$ and every path from v to u in G uses a bi-edge, replace G by $G - (u, v)$.*

Reduction 16 (DOME). *If there is an arc $(v, u) \in E(G)$ such that $(u, v) \notin E(G)$ and one of the following holds*
- $\{ p \mid (p, v) \in E(G), (v, p) \notin E(G) \} \subseteq \{ p \mid (p, u) \in E(G) \}$, *i.e., for every $(p, v) \in E(G)$ that is not a bi-edge there is an arc $(p, u) \in E(G)$.*

Algorithm 4 An algorithm that checks whether a vertex v is unconfined in a graph G.

1: **function** CHECKUNCONFINED(v, G)
2:     $S \leftarrow \{v\}$
3:     $P \leftarrow \{\, u \in N(S) \mid |N(u) \cap S| = 1 \,\}$
4:     **if** $P = \emptyset$ **then**
5:         $u \leftarrow \operatorname{argmin}_{u' \in P} |N(u') \setminus (N(S) \cup S)|$
6:         **if** $|N(u') \setminus (N(S) \cup S)| = 0$ **then return** True
7:         **else if** $|N(u) \setminus (N(S) \cup S)| = 1$ **then**
8:             $S \leftarrow S \cup \{u\}$
9:             **go to** 3
     return False

- $\{\, p \mid (u,p) \in E(G), (p,u) \notin E(G) \,\} \subseteq \{\, p \mid (v,p) \in E(G) \,\}$, *i.e., for every $(u,p) \in E(G)$ that is not a bi-edge there is an arc $(v,p) \in E(G)$.*

then replace G by $G - (v,u)$.

Reduction 17 (CORE)**.** *If there exists $v \in V(G)$ such that all arcs of v are bi-edges and the neighbor of v form a diclique, replace G by $G \circ v$.*

Note that also the CORE reduction is a special case of the INDICLIQUE and OUTDICLIQUE reductions.

B   Standard VCP Reductions

Reduction 18 (SUBSET [36])**.** *If there exists $v, u \in V(G)$ such that $\{v, u\} \in E(G)$ and $N(v) \subseteq N(u) \cup \{u\}$, then replace G by $G - u$.*

Another reduction by Fellows et al. is more complicated but generalizes many others like the 2FOLD reduction [37].

Reduction 19 (MANYFOLD [14])**.** *If there exists a vertex $v \in V(G)$ such that there is a partition (C_1, C_2) of $N(v)$, where*
- *$|C_1| \geq |C_2|$,*
- *C_i is a clique for $i = 1, 2$, and*
- *for each $c_1 \in C_1$, there is precisely one $c_2 \in C_2$ such that $\{c_1, c_2\} \notin E(G)$.*
Then, replace G by
$$G - v - C_2 + \bigcup_{\{c_1, c_2\} \in M, c_1 \in C_1} \{c_1\} \times N(c_2),$$
where M denotes the set of missing edges from $G[N(v)]$.

Whereas MANYFOLD works well on dense graphs, the following reduction works on sparse graphs.

Reduction 20 (4PATH [14])**.** *If there exists a vertex $v \in V(G)$ such that*
- *$N(v) = \{a, b, c, d\}$, and*
- *$E(G[N(v)]) = \{\{a,b\}, \{b,c\}, \{c,d\}\}$,*
then replace G by

$$G - v + \{\{a,c\}, \{a,d\}, \{b,d\}\}$$
$$+ \{a,b\} \times N(d) + \{c,d\} \times N(a).$$

While the above reductions all capture a fixed graph pattern, the following applicability of the following one is determined by an iterative procedure in Algorithm 4.

Reduction 21 (UNCONFINED [37, 2])**.** *If there is a vertex $v \in V(G)$ such that* CHECKUNCONFINED(v, G), *replace G by $G - v$.*

All of these reductions induce (many) boundary reductions.

C Proofs

Theorem C.1. *For every graph G such that $r = \langle H, H', B, c \rangle$ is applicable it holds that for every minimum vertex cover S of $G + H$ there is a minimum vertex cover S' of $G + H'$ such that*
- $|S| = |S'| + c$, *and*
- $S \cap V(G) = S' \cap V(G)$.

Proof. So let S be a minimum vertex cover of $G+H$ and $X = B \cap S$. We know that $VC(H-X) = VC(H' - X) + c$ and that $S_l = S \cap (V(H) \setminus B)$ is a minimal vertex cover of $H - X$. Therefore, $|S_l| = VC(H - X)$, which implies that there exists a vertex cover S'_l of $H' - X$ such that $|S_l| = |S'_l| + c$. Then, $S' = S \cap V(G) \cup S'_l$ is a vertex cover of $G + H'$ and it holds that $|S| = |S'| + c$ and $S \cap V(G) = S' \cap V(G)$.

It remains to show that S' is also minimum. Assume that there was another vertex cover C of strictly smaller cardinality. Then we can use the same steps as above to obtain a vertex cover C' of $G + H$ of strictly smaller cardinality than S. This is a contradiction, which implies that S' is a minimum vertex cover. $\quad\square$

Theorem C.2. *Let G be a digraph and $r = \langle H, H', B, c \rangle$ be a boundary VCP reduction and $\Pi(G) = G' + V(H) + E(H)$ such that r is applicable to G'. If every vertex $v \in V(H) \setminus B$ only has bi-edges in G and every arc $(u, w) \in E(G)$ is a bi-edge or $|\{u, w\} \cap B| \le 1$, then*

$$DFVS(G) = DFVS(G^*) + c,$$

where G^* is given by

$$G - (V(H) \setminus B) - B \times B$$
$$+ V(H') + \{ (u, v) \mid \{u, v\} \in E(H') \}.$$

Proof. The proof is analogous to that of Theorem 4.2. $\quad\square$

Theorem C.3. *Let G be a loop-free digraph, such that MANYFOLD is applicable to $v^* \in V(G)$ and G' be the graph obtained from G after MANYFOLD was applied on vertex v^*, then*
- $DFVS(G) = DFVS(G') + |C_2|$,
- *and given a minimum DFVS of G', we can in polynomial time compute a minimum DFVS of G.*

Proof. The main observation that is used to proof soundness for the MANYFOLD reduction in the VCP case, is that without loss of generality there are only two possible cases we need to consider. For this, first note that if v^* is not contained in a minimum DFVS of G, then the vertices in $C_1 \cup C_2 = N_{bi}^G(v^*)$ are. On the other hand, if v^* is in a minimum DFVS S of G, then still, since C_1 and C_2 are dicliques, it holds that $|S \cap C_i| \ge |C_i| - 1$ for $i = 1, 2$. In fact, if $|S \cap C_i| = |C_i|$ for one of $i = 1, 2$ then we can assume that it holds for both $i = 1$ and $i = 2$, since in this case v^* must be in S and we can replace it by the missing vertex without changing the size. Therefore, w.l.o.g. for a minimum DFVS S of G it holds that either

1. $S \cap C_i = C_i$ for $i = 1, 2$ and $v^* \notin S$, or

2. $|S \cap C_i| = |C_i| - 1$ for $i = 1, 2$.

19

Assume now, that we are given a minimum DFVS S of G such that 1. holds. In this case, we obtain a DFVS of G' as $S' = S \setminus C_2$ and it follows that $DFVS(G) - |C_2| \geq DFVS(G')$. To see that S' is a DFVS of G' observe that every edge that G' has but not G contains a vertex from $C_1 \subseteq S \setminus C_2 = S'$.

If instead we are given a minimum DFVS S of G such that 2. holds, then let $c_i \in C_i \setminus S$. In this case, $S' = S \setminus (C_2 \cup \{v^*\})$ is a DFVS of G' and thus $DFVS(G) - |C_2 \setminus \{c_2\}| + |\{v^*\}| = DFVS(G) - |C_2| \geq DFVS(G')$. To see that S' is a DFVS of G', assume the contrary. Then there must be cycle that is not covered by S'.

We know that at at least one of (c_1, c_2) and (c_2, c_1) is in M and proceed by a case distinction on whether both are in M or not.

Case both are in M: Then there is no uncovered path between c_1 and c_2 since the reduction is applicable. Assume there is a (w.l.o.g.) uncovered cycle. Then this cycle must use the vertex c_1, since the only added arcs, which do not have a vertex in S' use c_1. Furthermore, the cycle must contain an arc between $N^G(c_2)$ and c_1 and between $N^G(c_1)$ and c_1. If only the latter holds true, then the same cycle is also present in $G - S$. If only the former holds true, then there is an equivalent cycle in $G - S$, where c_1 is replaced by c_2. Since the cycle is uncovered, the arcs must go into opposite direction, i.e., be of the form (v_2, c_1) and (c_1, v_1) or (v_1, c_1) and (c_1, v_2), where $v_i \in N^G(c_i), i = 1, 2$. Assume the first form, then we can transform the uncovered cycle into an uncovered path from c_1 to c_2 in G by going from v_2 to c_2 instead of c_1. This is a contradiction to the assumption that there are no uncovered paths between c_1 and c_2. The argument for the latter form is analogous.

Case only (c_1, c_2) is in M. Thus, every uncovered path in G from c_2 to c_1 uses the arc (c_2, c_1). This implies that a cycle $c_1 \ldots c_1$ in $G' - S'$ is also a cycle in $G - S$, there is a corresponding cycle with c_1 replaced by c_2 in G, or there is a corresponding cycle $c_1 \ldots c_2 c_1$ in $G - S$. This a contradiction to the assumption that S is DFVS of G.

Case only (c_2, c_1) is in M. Thus, every uncovered path in G from c_1 to c_2 uses the arc (c_1, c_2). This implies that a cycle $c_1 \ldots c_1$ in $G' - S'$ is also a cycle in $G - S$, there is a corresponding cycle with c_1 replaced by c_2 in G, or there is a corresponding cycle $c_1 c_2 \ldots c_1$ in $G - S$. This a contradiction to the assumption that S is DFVS of G.

Thus, it follows that $DFVS(G) \geq DFVS(G') + |C_2|$.

As for the other direction, let S' be a minimum DFVS of G'. Again, we consider two cases, namely $|S' \cap C_1| = |C_1|$ and $|S' \cap C_1| = |C_1| - 1$, which are the only possible cases for the cardinality of the intersection.

Case $|S' \cap C_1| = |C_1|$: Then $S = S' \cup C_2$ is a DFVS of G, which implies $DFVS(G) \leq DFVS(G') + |C_2|$. To see that S is a DFVS of G, observe that $G - S - v^*$ and $G' - S'$ are equal and $N(v^*) \subseteq C_1 \cup C_2 \subseteq S$.

Case $|S' \cap C_1| = |C_1| - 1$: Let $c_1 \in C_1 \setminus S'$ and $c_2 \in C_2$ the unique vertex such that there is no bi-edge between c_1 and c_2 in G. Then $S = S' \cup (C_2 \setminus \{c_2\}) \cup \{v^*\}$ is a DFVS of G, which implies $DFVS(G) \leq DFVS(G') + |C_2|$. It remains to show that S is a DFVS of G. We consider two subcases for the number of arcs that use c_1 and c_2 in M.

Case only one of (c_1, c_2) is in M or (c_2, c_1) is in M. Assume (c_1, c_2) is in M, the other case works analogously. Thus, every uncovered path in G from c_2 to c_1 uses the arc (c_1, c_2). Assume that there is a (w.l.o.g.) uncovered cycle in $G - S$. We know that this cycle must use c_1 and is thus of the form $c_1 \ldots c_1$. Furthermore it must use $c2$, which implies that there is an uncovered path from c_2 to c_1 as a part of the cycle. This the cycle is actually of the form $c_1 \ldots c_2 c_1$. Then however, there is a corresponding cycle $c_1 \ldots c_1$ in $G' - S'$, since all the arcs of c_2 were added to c_1, which is a contradiction.

Case both (c_1, c_2) and (c_2, c_1) are in M. Assume that there is a (w.l.o.g.) uncovered cycle in $G - S$. As in the previous case, we know that the cycle must use both c_1 and c_2, since it is otherwise also a cycle in $G' - S'$. Thus, this cycle gives us an uncovered path from c_1 to c_2, which is a contradiction to the assumption on M.

Since these are the only cases, we are done and $DFVS(G) \leq DFVS(G') + |C_2|$, meaning

that overall $DFVS(G) = DFVS(G') + |C_2|$. Furthermore, the constructions used in the proof are possible in polynomial time, which was the second claim of the theorem. $\quad\square$

Theorem C.4. *Let G be a digraph such that 4PATH is applicable to $v^* \in V(G)$ and G' be the graph obtained from G after 4PATH was applied to the vertex v^*, then*
- *$DFVS(G) = DFVS(G')$, and*
- *given a minimum DFVS of G', we can in polynomial time compute a minimum DFVS of G.*

Proof. Let S be a minimum DFVS of G. If $N^G(v^*) \subseteq S$, then S is also a DFVS of G' and $DFVS(G) \geq DFVS(G')$, since every added arc uses a vertex from $N^G(v^*)$. Otherwise, we can assume w.l.o.g. that $|N^G(v^*) \cap S| = 2$ and $v \in S$, since for every minimum DFVS of G, with $|N^G(v^*) \cap S| = 3$, there is a minimum DFVS S^* with $N^G(v^*) \subseteq S^*$ of the same size. Furthermore, there cannot be a DFVS of G, which contains less than two elements from $N^G(v^*)$, since the elements in $N^G(v^*)$ form a path of four elements, where every arc is a bi-edge. So let S be a minimum DFVS of G such that $|N^G(v^*) \cap S| = 2$ and $v \in S$. We proceed by case distinction over the two neighbors of v^* that are in S.

Case $N(v^*) \cap S = \{b, c\}$: In this case $S' = (S \setminus \{v^*\}) \cup \{a\}$ is a DFVS of G' and $DFVS(G) \geq DFVS(G')$. Assume that on the contrary, there is a (w.l.o.g.) uncovered cycle. Then this cycle must use the vertex d, since the only added arcs, which do not have a vertex in S' use d. Furthermore, the cycle must contain an arc between $N^G(a)$ and d and between $N^G(d)$ and d. If only the latter holds true, then the same cycle is also present in $G - S$. If only the former holds true, then there is an equivalent cycle in $G - S$, where d is replaced by a. Since the cycle is uncovered, the arcs must go into opposite direction, i.e., be of the form (v_a, d) and (d, v_d) or (v_d, d) and (d, v_a). Assume the first form, then we can transform the uncovered cycle into an uncovered path from d to a in G by going from v_a to a instead of d. This is a contradiction to the assumption that there are no uncovered paths between a and d. The argument for the latter form is analogous.

Case $N(v^*) \cap S = \{a, c\}$: In this case $S' = (S \setminus \{v^*\}) \cup \{d\}$ is a DFVS of G' and $DFVS(G) \geq DFVS(G')$. The argument showing that S' is a DFVS of G' is analogous to that of the previous case.

Case $N(v^*) \cap S = \{b, d\}$: In this case $S' = (S \setminus \{v^*\}) \cup \{a\}$ is a DFVS of G' and $DFVS(G) \geq DFVS(G')$. The argument showing that S' is a DFVS of G' is analogous to that of the first case.

As for the other direction, let S' be a minimum DFVS of G'. If $N^G(v^*) \subseteq S'$, then S' is also a DFVS of G and $DFVS(G) \leq DFVS(G')$. Otherwise, we know that $|N^G(v^*) \cap S'| = 3$, since $G'[N^G(v^*)]$ is a diclique.

Case $N^G(v^*) \cap S' = \{a, b, c\}$: Then $S = (S' \setminus \{a\}) \cup \{v^*\}$ is a DFVS of G and $DFVS(G) \leq DFVS(G')$. Assume that on the contrary there is a cycle. Then this cycle must contain a. However, since d has the same neighbors as a in G' this implies the existence of an equivalent cycle with a replaced by d in G', which is a contradiction to $d \notin S'$.

Case $N^G(v^*) \cap S' = \{a, b, d\}$: Then $S = (S' \setminus \{a\}) \cup \{v^*\}$ is a DFVS of G and $DFVS(G) \leq DFVS(G')$ by an analogous argument as above.

The other two cases follow from symmetric arguments. $\quad\square$

In order to prove soundness of UNCONFINED, we use the following definitions inspired by [37].

For a set $A \subseteq V(G)$ such that $G[A]$ is acyclic let

$$N_c(A) = \{\, u \in V(G) \mid G[A \cup \{u\}] \text{ is cyclic} \,\},$$

i.e., the neighbors of any vertex in A such that there is a cycle through A and the vertex. A vertex $u \in N_c(A)$ is called a *directed child* of A if it has exactly two arcs that are shared with vertices in A (i.e., $|N_{succ}(u) \cap A| + |N_{pred}(u) \cap A| = 2$). The vertices in A that share arcs with u are called its *parents*.

Lemma C.1. *Let A be a set of vertices from G such that*
- *$G[A]$ is acyclic, and*
- *for every minimum DFVS S of G it holds that $A \cap S = \emptyset$.*

Then for each directed child u of A no minimum DFVS of G contains all vertices $w \in N(u) \setminus (N_c(A) \cup A)$.

Proof. Assume that there is a minimum DFVS S of G such that $S \cap (N(u) \setminus (N_c(A) \cup A)) = (N(u) \setminus (N_c(A) \cup A))$ for some directed child $u \in N_c(A)$. The parents p_1, p_2 of u are in A and thus by assumption not in S. Furthermore, since S is a DFVS and $A \cap S = \emptyset$, we know that $N_c(A) \subseteq S$ since $N_c(A)$ contains the vertices v such that $G[A \cup \{v\}]$ is cyclic. It follows that $N(u) \setminus A \subseteq S$ and thus $N(u) \setminus A \subseteq S'$, where $S' = (S \cup \{p_1\}) \setminus \{u\}$ (or $(S \cup \{p_2\}) \setminus \{u\}$). Recall that u is a directed child of A and therefore only has two arcs that go from or to A, which use the parents. Since we put one of u's parents into S' there is at most one arc that uses u in the graph $G - S'$, implying that u cannot be contained in any cycle in $G - S'$. Thus, the removal of u can be compensated by the addition of p_1 (or p_2) and we see that S' is a minimum DFVS of G, which shares a vertex with A. This is a contradiction. $\square$

Theorem C.5. *Let G be a digraph. After applying UNCONFINED to vertex v resulting in G', it holds that for every minimum DFVS S of G' the set $S \cup \{v\}$ is a minimum DFVS of G.*

Proof. The result follows from Lemma C.1. The idea of Algorithm 2 is the following: We start by assuming that $S = \{v\}$ has an empty intersection with any minimum DFVS. If this leads to a contradiction together with Lemma C.1 then v must be contained in some DFVS and the theorem follows.

In particular, we get a contradiction, when there is a directed child u such that $|N(u) \setminus (N_c(A) \cup S)| = 0$. In this case, we know that there is a DFVS of G that contains v. If there is no immediate contradiction but there is a $u \in N_c(A)$ with $|N(u) \setminus (N_c(A) \cup S)| = 1$, then we know that the unique vertex $v' \in N(u) \setminus (N_c(A) \cup S)$ must also not be contained in any minimum DFVS of G, which means that we can extend S by adding v'. $\square$

References

[1] Faisal N Abu-Khzam and Michael A Langston. A direct algorithm for the parameterized face cover problem. In *IWPEC 2004*, pages 213–222. Springer, 2004. URL `https://doi.org/10.1007/978-3-540-28639-4_19`.

[2] Takuya Akiba and Yoichi Iwata. Branch-and-reduce exponential/fpt algorithms in practice: A case study of vertex cover. *Theoretical Computer Science*, 609:211–225, 2016. URL `https://doi.org/10.1016/j.tcs.2015.09.023`.

[3] Gilles Audemard and Laurent Simon. Predicting learnt clauses quality in modern SAT solvers. In Craig Boutilier, editor, *IJCAI 2009*, pages 399–404, 2009. URL `http://ijcai.org/Proceedings/09/Papers/074.pdf`.

[4] Florent Avellaneda. A short description of the solver EvalMaxSAT. In *MaxSAT Evaluation 2020*, pages 8–9, 07 2020. URL `http://florent.avellaneda.free.fr/dl/EvalMaxSAT.pdf`.

[5] Yu Bao, Morihiro Hayashida, Pengyu Liu, Masayuki Ishitsuka, Jose C Nacher, and Tatsuya Akutsu. Analysis of critical and redundant vertices in controlling directed complex networks using feedback vertex sets. *Journal of Computational Biology*, 25(10):1071–1090, 2018. URL `https://doi.org/10.1089/cmb.2018.0019`.

[6] Armin Biere, Marijn Heule, Hans van Maaren, and Toby Walsh, editors. *Handbook of Satisfiability - Second Edition*, volume 336 of *Frontiers in Artificial Intelligence and Applications*. IOS Press, 2021. ISBN 978-1-64368-160-3. doi: 10.3233/FAIA336. URL `https://doi.org/10.3233/FAIA336`.

[7] Robert Brummayer and Armin Biere. Effective bit-width and under-approximation. In *EUROCAST 2009*, volume 5717 of *Lecture Notes in Computer Science*, pages 304–311. Springer, 2009. URL `https://doi.org/10.1007/978-3-642-04772-5_40`.

[8] Jianer Chen, Yang Liu, Songjian Lu, Barry O'sullivan, and Igor Razgon. A fixed-parameter algorithm for the directed feedback vertex set problem. In *STOC 2008*, pages 177–186, 2008. URL `https://doi.org/10.1145/1374376.1374404`.

[9] Edmund M. Clarke, Orna Grumberg, Somesh Jha, Yuan Lu, and Helmut Veith. Counterexample-guided abstraction refinement for symbolic model checking. *J. ACM*, 50 (5):752–794, 2003. URL `https://doi.org/10.1145/876638.876643`.

[10] Wolfgang Dvořák, Sebastian Ordyniak, and Stefan Szeider. Augmenting tractable fragments of abstract argumentation. *Artif. Intell.*, 186:157–173, 2012. URL `https://doi.org/10.1016/j.artint.2012.03.002`.

[11] Wolfgang Dvořák, Markus Hecher, Matthias König, André Schidler, Stefan Szeider, and Stefan Woltran. Tractable abstract argumentation via backdoor-treewidth. In *AAAI 2022*, pages 5608–5615. AAAI Press, 2022. URL `https://ojs.aaai.org/index.php/AAAI/article/view/20501`.

[12] Guy Even, B Schieber, M Sudan, et al. Approximating minimum feedback sets and multicuts in directed graphs. *Algorithmica*, 20(2):151–174, 1998. URL `https://doi.org/10.1007/PL00009191`.

[13] François Fages and Akash Lal. A constraint programming approach to cutset problems. *Computers & Operations Research*, 33(10):2852–2865, 2006. URL `https://doi.org/10.1016/j.cor.2005.01.014`.

[14] Michael R. Fellows, Lars Jaffke, Aliz Izabella Király, Frances A. Rosamond, and Mathias Weller. What is known about vertex cover kernelization? In *Adventures Between Lower Bounds and Higher Altitudes - Essays Dedicated to Juraj Hromkovič on the Occasion of His 60th Birthday*, volume 11011 of *Lecture Notes in Computer Science*, pages 330–356. Springer, 2018. doi: 10.1007/978-3-319-98355-4_19. URL `https://doi.org/10.1007/978-3-319-98355-4_19`.

[15] Rudolf Fleischer, Xi Wu, and Liwei Yuan. Experimental study of fpt algorithms for the directed feedback vertex set problem. In *ESA 2009*, pages 611–622. Springer, 2009. URL `https://doi.org/10.1007/978-3-642-04128-0_55`.

[16] Meinrad Funke and Gerhard Reinelt. A polyhedral approach to the feedback vertex set problem. In *IPCO 1996*, pages 445–459. Springer, 1996. URL `https://doi.org/10.1007/3-540-61310-2_33`.

[17] Gael Glorian, Jean-Marie Lagniez, Valentin Montmirail, and Nicolas Szczepanski. An incremental sat-based approach to the graph colouring problem. In *CP 2019*, volume 11802 of *Lecture Notes in Computer Science*, pages 213–231. Springer, 2019. URL `https://doi.org/10.1007/978-3-030-30048-7_13`.

[18] Mikolás Janota, Radu Grigore, and João Marques-Silva. Counterexample guided abstraction refinement algorithm for propositional circumscription. In Tomi Janhunen and Ilkka Niemelä, editors, *Logics in Artificial Intelligence - 12th European Conference, JELIA 2010, Helsinki, Finland, September 13-15, 2010. Proceedings*, volume 6341 of *Lecture Notes in Computer Science*, pages 195–207. Springer, 2010. URL `https://doi.org/10.1007/978-3-642-15675-5_18`.

[19] Mikolás Janota, William Klieber, João Marques-Silva, and Edmund M. Clarke. Solving QBF with counterexample guided refinement. *Artif. Intell.*, 234:1–25, 2016. URL `https://doi.org/10.1016/j.artint.2016.01.004`.

[20] Mikolás Janota, Radu Grigore, and Vasco M. Manquinho. On the quest for an acyclic graph. In *RCRA@AI*IA 2017*, volume 2011 of *CEUR Workshop Proceedings*, pages 33–44. CEUR-WS.org, 2017. URL `http://ceur-ws.org/Vol-2011/paper4.pdf`.

[21] Richard M. Karp. Reducibility among combinatorial problems. In *Complexity of Computer Computations 1972*, The IBM Research Symposia Series, pages 85–103. Plenum Press, New York, 1972. doi: 10.1007/978-1-4684-2001-2_9. URL `https://doi.org/10.1007/978-1-4684-2001-2_9`.

[22] Markus Kirchweger and Stefan Szeider. SAT modulo symmetries for graph generation. In Laurent D. Michel, editor, *27th International Conference on Principles and Practice of Constraint Programming, CP 2021, Montpellier, France (Virtual Conference), October 25-29, 2021*, volume 210 of *LIPIcs*, pages 34:1–34:16. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2021. doi: 10.4230/LIPIcs.CP.2021.34. URL `https://doi.org/10.4230/LIPIcs.CP.2021.34`.

[23] Henning Koehler. A contraction algorithm for finding minimal feedback sets. In *ACSC 38*, pages 165–173, 2005. URL `http://crpit.scem.westernsydney.edu.au/abstracts/CRPITV38Koehler.html`.

[24] Jean-Marie Lagniez, Daniel Le Berre, Tiago de Lima, and Valentin Montmirail. A recursive shortcut for CEGAR: application to the modal logic K satisfiability problem. In Carles Sierra, editor, *Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence, IJCAI 2017, Melbourne, Australia, August 19-25, 2017*, pages 674–680. ijcai.org, 2017. doi: 10.24963/ijcai.2017/94. URL `https://doi.org/10.24963/ijcai.2017/94`.

[25] Mile Lemaic. *Markov-Chain-Based Heuristics for the Feedback Vertex Set Problem for Digraphs*. PhD thesis, Universität zu Köln, 2008. URL `https://kups.ub.uni-koeln.de/2547/`.

[26] Hanoch Levy and David W Low. A contraction algorithm for finding small cycle cutsets. *Journal of algorithms*, 9(4):470–493, 1988. URL `https://doi.org/10.1016/0196-6774(88)90013-2`.

[27] Hen-Ming Lin and Jing-Yang Jou. On computing the minimum feedback vertex set of a directed graph by contraction operations. *ICCD 1999*, 19(3):295–307, 1999. URL `https://doi.org/10.1109/ICCD.1999.808567`.

[28] António Morgado, Carmine Dodaro, and João Marques-Silva. Core-guided MaxSAT with soft cardinality constraints. In Barry O'Sullivan, editor, *Principles and Practice of Constraint Programming - 20th International Conference, CP 2014, Lyon, France, September 8-12, 2014. Proceedings*, volume 8656 of *Lecture Notes in Computer Science*, pages 564–573. Springer, 2014. doi: 10.1007/978-3-319-10428-7_41. URL `https://doi.org/10.1007/978-3-319-10428-7_41`.

[29] Matthew W. Moskewicz, Conor F. Madigan, Ying Zhao, Lintao Zhang, and Sharad Malik. Chaff: Engineering an efficient SAT solver. In *Proceedings of the 38th Design Automation Conference, DAC 2001, Las Vegas, NV, USA, June 18-22, 2001*, pages 530–535. ACM, 2001. URL https://doi.org/10.1145/378239.379017.

[30] George L Nemhauser and Leslie Earl Trotter. Vertex packings: structural properties and algorithms. *Mathematical Programming*, 8(1):232–248, 1975. URL https://doi.org/10.1007/BF01580444.

[31] Christian Schulz, Ernestine Großmann, Tobias Heuer, and Darren Strash. Pace 2022, 2022. URL https://pacechallenge.org/2022/.

[32] Jendrik Seipp and Malte Helmert. Counterexample-guided cartesian abstraction refinement for classical planning. *J. Artif. Intell. Res.*, 62:535–577, 2018. doi: 10.1613/jair.1.11217. URL https://doi.org/10.1613/jair.1.11217.

[33] Abraham Silberschatz, Peter Baer Galvin, and Greg Gagne. *Operating System Concepts, 10th Edition*. Wiley, 2018. ISBN 978-1-118-06333-0. URL http://os-book.com/OS10/index.html.

[34] João P. Marques Silva and Karem A. Sakallah. GRASP: A search algorithm for propositional satisfiability. *IEEE Trans. Computers*, 48(5):506–521, 1999. URL https://doi.org/10.1109/12.769433.

[35] Takehide Soh, Daniel Le Berre, Stéphanie Roussel, Mutsunori Banbara, and Naoyuki Tamura. Incremental sat-based method with native boolean cardinality handling for the hamiltonian cycle problem. In *JELIA 2014*, volume 8761 of *Lecture Notes in Computer Science*, pages 684–693. Springer, 2014. URL https://doi.org/10.1007/978-3-319-11558-0_52.

[36] Ulrike Stege and Michael Ralph Fellows. An improved fixed parameter tractable algorithm for vertex cover. *Technical report/Departement Informatik, ETH Zürich*, 318, 1999. URL https://www.research-collection.ethz.ch/bitstream/handle/20.500.11850/69332/eth-4359-01.pdf.

[37] Mingyu Xiao and Hiroshi Nagamochi. Confining sets and avoiding bottleneck cases: A simple maximum independent set algorithm in degree-3 graphs. *Theor. Comput. Sci.*, 469:92–104, 2013. doi: 10.1016/j.tcs.2012.09.022. URL https://doi.org/10.1016/j.tcs.2012.09.022.

[38] Hai-Jun Zhou. A spin glass approach to the directed feedback vertex set problem. *Journal of Statistical Mechanics: Theory and Experiment*, 2016(7):073303, 2016. URL https://doi.org/10.1088/1742-5468/2016/07/073303.